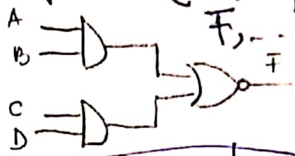
library NumeLib, ... ;
use NumeSublib ... ;
entity NumeEnt is
port (A, B, ... : in Type;
F, ... : out Type
architecture NumeA of NEns
begin <= ;
end NumeA;

2. ENTITY DESIGN

1. ENTITY ======================

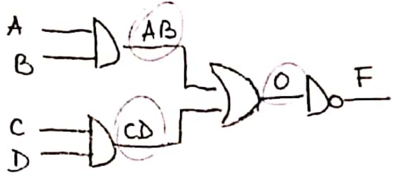
```
library IEEE;
    use IEEE.STD_LOGIC_1164.all;

entity AOI is
    port (A, B, C, D: in STD_LOGIC;
        F: out STD_LOGIC);
end AOI;

architecture V1 of AOI is
begin
    F <= not((A and B) or (C and D));
end V1;
```



2. SIGNAL ---------------------------- var. în înăuntrul (sunt atât in, cât și out)

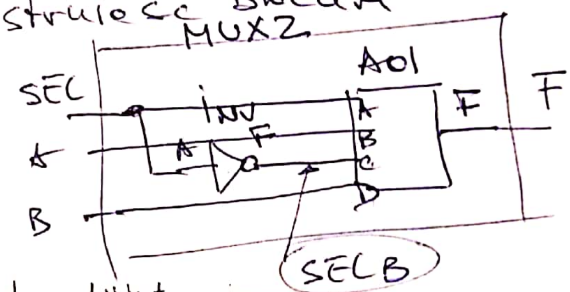
```
architecture V2 of AOI is
    signal AB, CD, O: STD_LOGIC;
begin
    AB <= A and B after 2 NS;
    CD <= C and D after 2 NS;
    O <= AB or CD after 2 NS;
    F <= not O after 1 NS;
end V2;
```

întârziere de 2ns



3. COMPONENT =================

construiesc blocuri noi



New

```
library IEEE;
    use IEEE.STD_LOGIC_1164.all;

entity MUX2 is
    port (SEL, A, B: in STD_LOGIC;
        F: out STD_LOGIC);
end MUX2;

architecture STRUCTURE of MUX2 is
    component INV
        port (A: in  STD_LOGIC;
            F: out STD_LOGIC);
    end component;

    component AOI
        port (A, B, C, D: in STD_LOGIC;
            F: out STD_LOGIC);
        end component;

    signal SELB: STD_LOGIC;
begin
    G1: INV port map(SEL, SELB);
    G2: AOI port map(SEL, A, SELB, B, F);
end STRUCTURE;
```

se găsește în biblioteca IEEE

declarată identic ca la entity

legături interne

câte comp. am atâtea port map-uri trebuie să fie

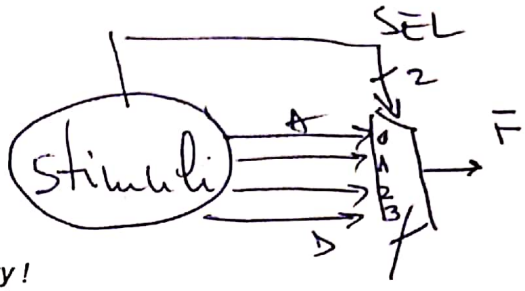
Et: NumeC port map (listă legături) MUX2
{ port Comp( ) → { in/out EntNew
signals

in/out ale noi funcțiuni, sau semnale !

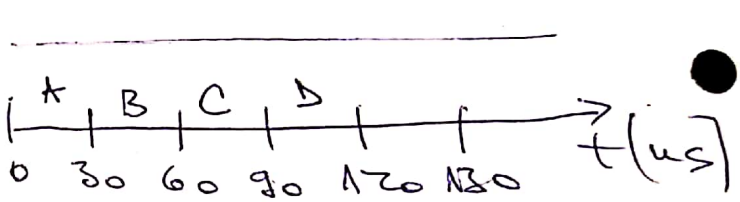
4. --

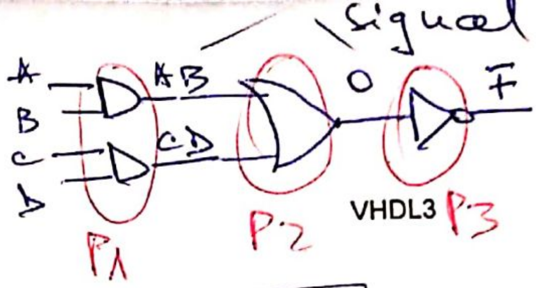
```
G1: entity WORK.INV(ARCH)
    port map (SEL, SELB);
G2: entity WORK.AOI(V2)
    port map (SEL, A, SELB, B, F);
```

Nr. elem. lista port map = Nr. elem. lista comp.




```
5. TEST BENCHES ===============
library IEEE;
    use IEEE.STD_LOGIC_1164.all;
(or  use WORK.all;)

entity TEST_MUX4 is
end;                                    -- empty entity !

architecture BENCH of TEST_MUX4 is
    component MUX4
        port (SEL: in STD_LOGIC_VECTOR(1 downto 0);
              A, B, C, D: in STD_LOGIC;
              F: out STD_LOGIC);
    end component;

    signal SEL: STD_LOGIC_VECTOR(1 downto 0);
    signal A, B, C, D, F: STD_LOGIC;

begin
    SEL <= "00",
           "01" after 30 NS,
           "10" after 60 NS, "11" after 90 NS,
           "XX" after 120 NS, "00" after 130 NS;
    A <= 'X',
         '0' after 10 NS,
         '1' after 20 NS;
    B <= 'X',
         '0' after 40 NS,
         '1' after 50 NS;
    C <= 'X',
         '0' after 70 NS,
         '1' after 80 NS;
    D <= 'X',
         '0' after 100 NS,
         '1' after 110 NS;
    ...
    M: MUX4 port map(SEL, A, B, C, D, F);

end BENCH;

configuration CFG_MUX4 of TEST_MUX4 is
    for BENCH
    end for;
end CFG_MUX4;
================================
```





3. PROCESS & SYNCHRONISATION

1. PROCESS ====================

```
architecture V3 of AOI is

    signal AB, CD, O: STD_LOGIC;

begin
    process (A, B, C, D)          -- sensitivity list !
    begin
        AB <= A and B after 2 NS;
        CD <= C and D after 2 NS;
    end process;

    process (AB, CD)              -- procese concurente !
    begin
        O <= AB or CD after 2 NS;
    end process;

    process (O)
    begin
        F <= not O after 1 NS;
    end process;

end V3;
```

procese cu LS

intrările — LS — în cadrul procesului – secvențialitate

1) procese concurente! 2) corp secvențial
3) se trezește dacă un elem. din LS schimbă de stare = Event
4) se exec. corp. → end
= comunicare → prin signal

2. SENSITIVITY LIST ----------------------

```
(P2: ) process (SEL, A, B, C)
begin
    if SEL = '1' then
        OP <= A and MASK;
    else
        OP <= B;
    end if;
end process (P2);
```

C — variabilă de sincronizare

procesul nu se rulează dacă MASK se modifică, pt. că nu e pus în cadrul procesului;

C — ∄ în corp proces
MASK — ∄ în LS

3. WAIT --

```
Stimulus: process
begin
    Reset <= '0';
    wait for 50 NS;
    Reset <= '1';
    wait;
end process;
```

begin → începe la t=0.
wait; → nu se atinge end process

procese fără LS
1) procese concurente, corp secvențial
2) Pornesc la start simulare (t=0).
3) wait [for x NS]
4) → end;

4. TEST VECTORS ----------------------

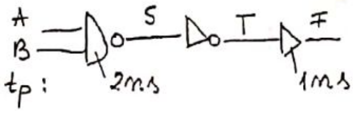
```
TestVectors: process
begin
    A <= "0000";
    B <= "0000";
    wait for 10 NS;
    A <= "1111";
    wait for 10 NS;
    B <= "1111";
    wait for 10 NS;
    A <= "0101";
    B <= "1010";
    wait;
end process;
```

process → fără listă

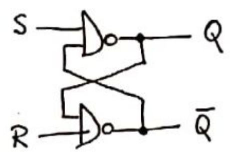
se rulează prima dată procesului.
– " – a II-a – " –

delta delay

```
5. EVENTS -----------------------------------
process (A, B, S, T)                      -- delta delay !
begin
    S <= A nand B after 2 NS;
    T <= not S;
    F <= T after 1 NS;
end process;
```




```
6. ------------------------------------------------
Feedback: process (S, R, Q, QB)        -- RS
begin
    Q  <= S nand QB;   – reacție
    QB <= R nand Q;
end process;
=================================
```



stare interz.

S	R	Q	$\bar{Q}$
0	0	1	1
0	1	1	0
1	0	0	1
1	1		

= memorează Q_a → Q anterior

(Ex. 5)

init. AB → "11" → S = 0, T = 1, F = 1

(1) A → 0 (LS) ⌐0 , { setez ev. la (S), execuție → end. | Nu se modifică

~~după 2S → S = 1~~

(2) după 2 Ns → S = 1, (LS) → { setez ev. la T, execuție → end

(3) după Δ delay → T = 0 (LS) → { setez ev. F, execuție → end

după 1 NS F = 0 (∄ LS)

⇒ Pt. calcule înlănțuite în LS ∃ variabile intermediare NU – ultima